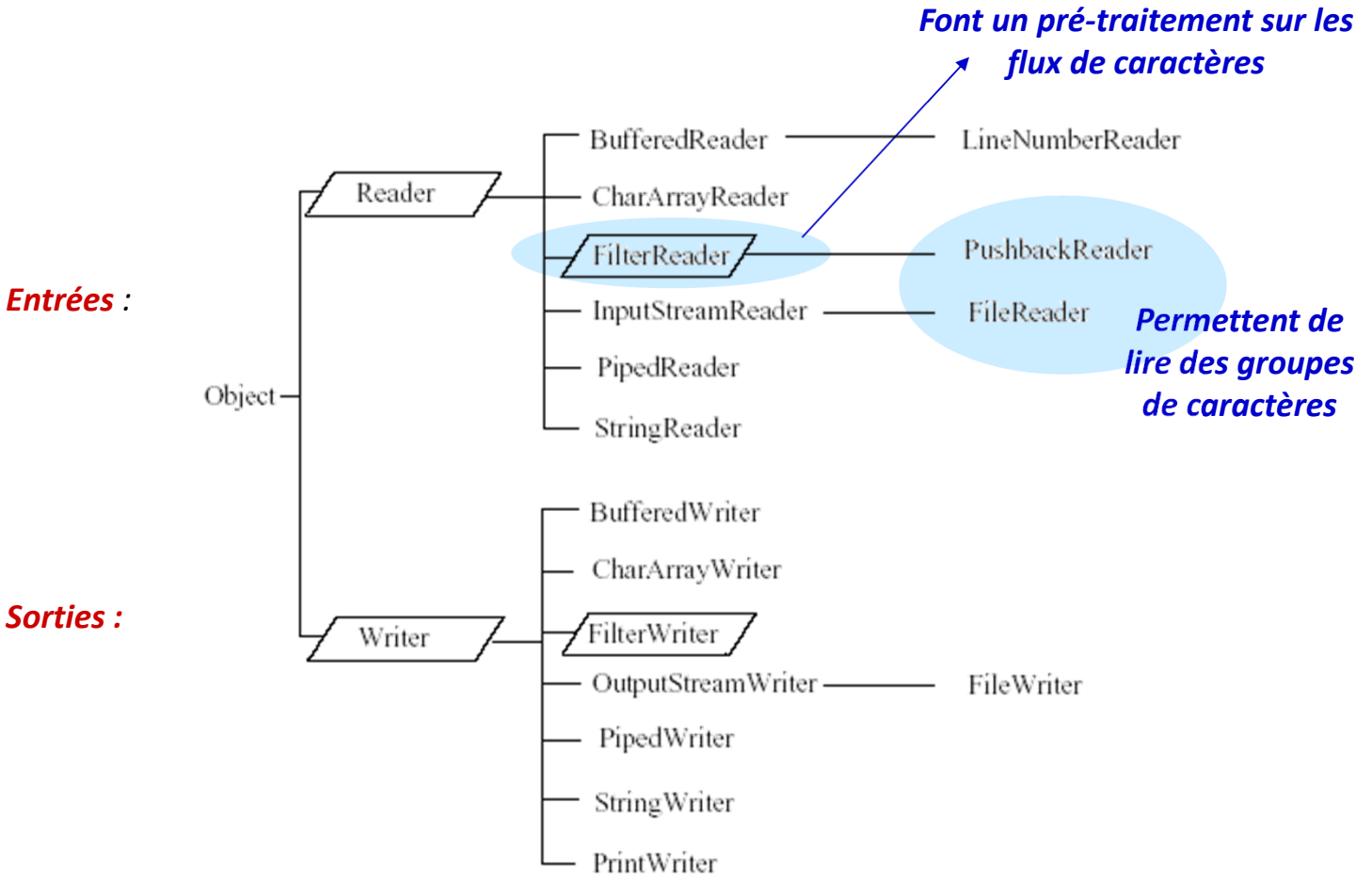
Etape 5

- 1.2.4 Flux caractères

Flux de caractères : Reader/Writer



int read() : lit le prochain caractère sur le flux et le retourne sous forme d'int. Pour l'utiliser réellement sous forme de caractère (char), il sera nécessaire de faire une conversion explicite (cast).

Types de flux caractères

- *FilterReader* : classe abstraite (méthodes pas défaut à surcharger) permettant de lire des flux de données filtrées.
- ***InputStreamReader*** : permet la communication entre les flux d'entrée d'octets et les flux d'entrée de caractères. Il va lire les octets et les transformer en caractères en utilisant un jeu de caractères spécifique (Charset). Ce jeu de caractères sera celui de la plateforme par défaut. Il peut être spécifié par l'utilisateur à l'instanciation : *InputStreamReader(InputStream in, Charset cs)*.
- *StringReader* : cette classe est un flux de caractères créé à partir d'une chaîne de caractères. Cette chaîne de caractères va être fournie lors de l'instanciation à l'aide du constructeur StringReader(String s).
- *CharArrayReader* : cette classe implémente une mémoire tampon (buffer) de caractères et peut être utilisée comme flux de caractères d'entrée.
- *BufferedReader* : permet de lire des flux de caractères stockés dans un buffer. Si la taille n'a pas été spécifiée (lors de l'instanciation) elle sera affectée par défaut ; Deux constructeurs *BufferedReader(Reader in), BufferedReader(Reader in, int taille)*

exemple d'utilisation

Exemple de manipulation de flux caractère en lecture :

```
import java.io.*;

public class TestDeReaderEnEntree
{
   public static void main(String[] args)
   {
      BufferedReader entree = new BufferedReader( new InputStreamReader(System.in));

        try{
              String ligne=entree.readLine();
        }

        catch (Exception e) {System.out.println(e.toString());}
}
}
```

Les flux analysés (parsés) : Stream Tokenizer

- Principe général : découper un flux entrant en tokens ;
- On lit des caractères, le constructeur utilise un Reader :

  StreamTokenizer lec= new StreamTokenizer(new InputStreamReader(System.in));
- On peut définir des séparateurs, des caractères à ignorer, …

  blancs, ponctuations, / ou //
- La classe StringTokenizer fait un travail proche sur les chaînes.

La méthode *nextToken()* retourne le type de l'unité lexicale suivante, type qui est caractérisé par une constante entière :

- ***TT_NUMBER*** **si l'unité lexicale représente un nombre. Ce nombre se trouve alors dans le champ nval (de type double), de l'instance de StreamTokenizer.**
- ***TT_WORD*** **si l'unité lexicale représente une chaîne de caractères. Cette chaîne se trouve alors dans le champ sval de l'instance du StreamTokenizer.**
- ***TT_EOL*** **si si l'unité lexicale représente une fin de ligne. La fin de ligne n'est reconnue comme unité lexicale que si on a utilisé l'instruction**

  nomDuStreamTokenizer.eolIsSignificant(true);
- ***TT_EOF*** **s'il s'agit du signe de fin de fichier**

Les flux analysés (parsés) : Stream Tokenizer

- On peut récupérer le type du *token* la variable d'instance *ttype*, la valeur numérique dans *nval* et la valeur de la String dans *sval*

```
StreamTokenizer lec= new StreamTokenizer( new
InputStreamReader(System.in));


while(! fin) {
     i = lec.nextToken();
      switch (lec.ttype) {
     case StreamTokenizer.TT_EOF: fin=true;break;

     case StreamTokenizer.TT_EOL: finDeLigne=true;break;

     case StreamTokenizer.TT_NUMBER :  i=(int) lec.nval;
                  S.o.p ("lu nombre:" + i); break;
     case StreamTokenizer.TT_WORD:    S.o.p ("lu mot:"+lec.sval); break;
     default : System.out.println ("lu autre:" +(char)i);
     }
} ….
```